PATENT APPLICATION
ATTY. DOCKET NO. 00100.02.0050

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

FILING OF A UNITED STATES PATENT APPLICATION

METHOD AND APPARATUS FOR DUAL PASS ADAPTIVE TESSELLATION

INVENTORS:

Vineet Goel 519 Dunraven Drive Winter Park, Florida 32792	Stephen Morein 10 Magazine Road Cambridge, Massachusetts 02139
Robert Scott Hartog 3326 Just A Mere Ct. Windermere, Florida 34786	

ASSIGNEE:

ATI Technologies Inc.
1 Commerce Valley Drive East
Markham, Ontario
Canada L3T 7X6

ATTORNEY OF RECORD:
TIMOTHY J. BECHEN
REGISTRATION NO. 48,126
VEDDER, PRICE, KAUFMAN & KAMMHOLZ, P.C.
222 NORTH LASALLE STREET, SUITE 2600
CHICAGO, ILLINOIS 60601
PHONE (312) 609-7500
FAX (312) 609-5005

Express Mail Label No. EL961994565US

Date of Deposit: March 2, 2004

I hereby certify that this paper is being deposited with the U.S. Postal Service "Express Mail Post Office to Addressee" service under 37 C.F.R. Section 1.10 on the date of deposit, indicated above, and is addressed to: Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313.

Name of Depositor: Christine A. Wright

Signature: Christine A. Wright

METHOD AND APPARATUS FOR DUAL PASS ADAPTIVE TESSELLATION

FIELD OF THE INVENTION

[0001] The present invention relates generally to graphics rendering and more specifically to the generation of standardized tessellation factors for rendering an image.

BACKGROUND OF THE INVENTION

[0002] In a graphics processing system, objects to be displayed are generally represented by a collection of polygons. Polygons are generally chosen due to the existence of efficient algorithms for the rendering of. However, frequently the object that is approximated by polygons is really a curved shape. Most methods for describing these surfaces fall under the classification of "higher order surfaces (HOS).

[0003] Patches, such as a Bezier patch, may represent a surface of an object. These are one representative type of HOS. These patches may typically contain information beyond standard pixel information and may further include information relating to surface normals, lighting source orientation, surface color and opacity and coordinates of a texture image to be mapped onto the surface. Sub-dividing the patches until the sub-patches are sufficiently flat so they may be approximated by a quadrilateral may be used to render the HOS or any other criterion to subdivide patches to render. The sub-patches may then be divided into triangles for graphics processing.

[0004] Another problem that occurs is the relative location of the object, such as a primitive, within a viewable output screen. The primitive, such as a triangle, defines an area include a plurality of pixels and due to the size of the primitive and the location of the primitive within a rendered scene, the ratio of the number of pixels relative to the primitive may be skewed.

[0005] As primitive may be deeper within the rendered scene, such as having a smaller depth value, the ratio of pixels per primitive may be reduced. During the rendering of the pixels, the depth offset may adversely effect the computation of tessellation factors. Using an adaptive tessellation technique, various adaptive tessellation factors may be computed using a software algorithm. Although, the technique includes processing time limitations and further includes space limitations as there must be physical space within a graphics processing system for a processor, such as a general purpose processor, executing operating instructions for calculating the adaptive tessellation factors.

[0006] Therefore, during the processing of tessellation factors, this technique may be inefficient as requiring the offloading of vertex information, the computation of the adaptive tessellation factors and the loading these tessellation factors back into the processing pipeline.

[0007] Another approach utilized for generating tessellation factors is an independent hardware configuration, which interfaces with the graphics rendering pipeline. This hardware approach provides a separate processor in communication with the graphics rendering pipeline, wherein this approach not only requires valuable real-estate for the extra hardware, but includes further processing time for the transmission of primitive indices to the hardware device and the transmission of vertex tessellated data back to the processing pipeline.

[0008] During the generation of tessellation factors, there are also limitations regarding the adaptability of the tessellation engine to perform discrete tessellation, continuous tessellation and adaptive tessellation. Current solutions provide for a tessellation engine to be designated for performing a predetermined type of tessellation, thereby only allowing for the reception of limited primitive types and thereby limited tessellation patterns.

[0009] As such, there exists a need for generating tessellation factors independent of the primitive types, wherein the tessellation factor generation efficiently uses existing available processing resources and without adversely effecting system processing speeds.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The present invention will be more readily understood with reference to the following drawings, wherein:

[0011] FIG. 1 illustrates an apparatus for dual pass adaptive tessellation as configured during a first pass in accordance with one embodiment of the present invention;

[0012] FIG. 2 illustrates the apparatus for dual pass adaptive tessellation as configured during a second pass in accordance with one embodiment of the present invention;

[0013] FIG. 3 illustrates a memory storing tessellation and index information in accordance with one embodiment of the present invention;

[0014] FIG. 4 illustrates a flow chart of the steps of a method for dual pass adaptive tessellation in accordance with one embodiment of the present invention;

[0015] FIG. 5 illustrates a flow chart of the steps of another method for dual pass adaptive tessellation in accordance with another embodiment of the present invention; and

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

[0016] FIG 1 illustrates an apparatus 100 for dual pass adaptive tessellation. The apparatus 100 includes a memory interface 102, a global register bus 104, a vertex grouper tessellator 106 having a direct memory access engine 108 dispose therein. The apparatus 100 further includes a shader processing unit 110, a plurality of vertex shader input staging registers 112 coupled to a plurality of vertex shaders 114 and a memory 116 coupled to the shaders 114 across a bus 118. The shader processing unit may be a sequencer or a control flow processor.

Moreover, each of the shaders 114 includes a math processing unit., which may be any suitable processing unit capable of executing mathematical operations.

[0017] In one embodiment, the vertex grouper 106 is pass-through enabled for a first pass. During the first pass, an index list 120 is provided from the memory interface 102 to the direct memory access engine 108. Furthermore, vertex grouper tessellator state data, direct memory access request data and a draw initiator 122 are received from the global register bus 104 by the vertex grouper tessellator 106. The indices of primitives are sent through the vertex grouper tessellator 106 to be loaded into registers of shader pipes, such as registers 112 and shaders 114. The vertex grouper tessellator 106 also sends an auto-index value for each primitive to one of the shader registers 112. Most specifically, the vertex grouper tessellator 106 provides primitive information 124 to the shader sequence 110 which thereupon distributes partitioned information 126 to each of the vertex shader input state registers 112.

[0018] The vertex grouper tessellator 106, during a first pass, also generates a per-process vector 130, a per primitive data 132 and a per packet data 134. During the first pass, no pixels are generated, but the signals 130-134 may still have to pass synchronization signals for vertex and state deallocation.

[0019] Referring back to the shader pipeline, the vertex shaders 114 receive vertex data arrays 134 retrieved from the memory 116. The shader processing unit 110 fetches corner vertices of high order surface primitive and computes tessellation factors based on corner vertices of edges of primitives. In another embodiment, during the first pass, the vertex shaders 114 could also generate control points for high order surfaces. The tessellator 106 is disabled during this pass.

[0020] Each of the vertex shaders 114 thereupon provides these computed values. In one embodiment, original indices 136 are written to the memory 116 across the bus 118 at indexed location as specified by auto-index value. In one embodiment, the control points and tessellation factors are written in separate linear buffers.

[0021] During the second pass, more than one indices can be passed to the vertex grouper tessellator 106 along with primitive types such as tri-list, quad-list, line-list, rect-patch, line-patch, or tri-patch. The vertex grouper tessellator 106, generates the parametric coordinance (u, v) for tensor products surfaces or bary-centric coordinates (u, v, w) for triangular surfaces using tessellation factors computed during first pass. These coordinates, along with original indices and/or auto-index value, are passed to the shader register 112 for evaluating tessellating vertices based on user evaluation shaders. The tessellation engine 100 also generates sub-primitive information for clipper and primitive assemblers for paths 134, 132, 130.

[0022] Fig. 2 illustrates the apparatus 100 through a second pass, with the vertex grouper tessellator 106 being enabled. The memory interface 102 receives a plurality of tessellation factors 150 received from the memory 116 of Fig. 1. Using the direct memory access engine 108 within the vertex grouper tessellator 106, the tessellation factors are retrieved by a direct memory access request 152, wherein the tessellation factor are provided for the vertex grouper tessellator 106 operating with an adaptive mode. In this second pass, the tessellation factors 150 are accessed as indices and an auto index generated, wherein the auto index 124 is loaded to the registers 112 of the shaders 114. A primitive mode indicator set to a tessellation mode such that the indices that go to the tessellation engine are interpreted as tessellation factors. The vertex group tessellator 106 generates bary-centric or tensor coordinates which are loaded to the shader registers 112 along with the auto-index value. The shaders 114 compute the tessellated vertices

by fetching control points from locations specified by the auto-index value and using bary centric or tensor product coordinates.

[0023] In one embodiment, a primitive data array 154 is retrieved from memory such that the vertex shaders 114 be utilized for the generation of newly computed sub-primitive information 156.

[0024] The vertex grouper tessellator 106 further generates the vector information 130, primitive information 132 and packet information 134. The per process vector information 130 includes state context vector and process vector size information. The per primitive information 132 includes sub-primitive assembly data including internal vertex indices, end of packet information and de-allocation information. The per packets information 134 includes state context vector and a sub-primitive type information.

[0025] Therefore, by using a dual-pass system for adaptive tessellation, the vertex grouper tessellator 106 is disabled in a pass-through mode for generating tessellation factors using the shaders 114 during pass one and the tessellator part of vertex grouper tessellator 106 is enabled in a tessellator mode for the second pass such that the tessellation factors themselves are used to generate tessellation outputs such as the per process vector output, the per primitive output and the per packet output, in conjunction with the sub primitive data 156. As such, the present invention utilizes a single tessellation engine 100 and reuses the same engine with different primitive types to generate the needed output. Therefore, present invention not only reduces the amount of real estate needed for hardware components, the engine 100 improves processing efficiency by allowing for the generation of tessellation output within the pixel/vertex processing pipeline.

[0026] The tessellation engine also generates vertices based on a vertex reuse number available to it, therefore providing for a scalable design. In one embodiment, a vertex reuse number may be a data indicator indicating the availability reutilizing vertex information which may be stored within a internal memory. The present invention allows for a compact design of the tessellation engine, a form of programmable tessellation where evaluation part can be specified by an end user. The present invention further includes a large range of primitive types per tessellation and a form of device capable of allowing data explosion by generating more vertices in providing sub primitive information.

[0027] In one embodiment, the tessellation engine 100 is a fixed-function block. This engine 100 understands a finite set of modes, each of which is defined for a specific purpose. The below table shows all of the modes supported by the present invention. As recognized by one having ordinary skill in the art, the below table indicates various modes for exemplary purposes only and illustrates one embodiment of the present invention. As recognized by one having ordinary skill in the art, any other suitable implementation in accordance with the information listed below for adaptive tessellation may also be suitably implemented and within the scope of the present invention.

Mode Name	Tesselator Input					Tesselator Output		VTG Output Prim Type	VGT Output (all floats)
	Prim Type (from PG)	Index Memory Size	Two Cycle Input Mode	Cycle 0	Cycle 1	Two Cycle Output Mode	Tess Output (all fixed)		
Line List – Single	2	std	0	2 std indx	--	0	2 std indx	Line List	2 indx 1 coord
Tri List – Single	4	std	0	3 std indx	--	1	3 std indx	Tri List	3 indx 3 coord
Quad List – Single	13	std	1	3 std indx	1 std indx	1	4 std indx	Tri List	4 indx 2 coord

L-Patch – Single	24	std	0	1 std indx	--	0	1 std indx	Line List	1 std indx 1 coord
T-Patch – Single Pass	25	std	0	1 std indx	--	1	1 std indx 1 quad	Tri List	1 std indx 1 quad id 3 coord
R-Patch – Single Pass	26	std	0	1 std indx	--	1	1 std indx 1 quad	Tri List	1 std indx 1 quad id 2 coord
Last of Multi-pass	24	--	0	1 auto-index	--	0	1 auto-index 1 quad	Line List	1 auto 1 quad id 1 coord
Last of Multi-pass	24	1 tess factor (float)	0	1 tess factor (float)	--	0	1 auto-index 1 quad		1 auto 1 quad id 1 coord
Last of Multi-pass	25	--	0	1 auto-index	--	1	1 auto-index 1 quad	Tri List	1 auto 1 quad id 3 coord
Last of Multi-pass	25	3 tess factors (float)	1	3 tess factors (float)	1 auto-index	1	1 auto-index 1 quad		1 auto 1 quad id 2 coord
Last of Multi-pass	26	--	0	1 auto-index	--	1	1 auto-index 1 quad	Tri List	1 auto 1 quad id 2 coord
Last of Multi-pass	26	4 tess factors (float)	1	3 tess factors (float)	1 tess factor 1	1	1 auto-index 1 quad		1 auto 1 quad id 2 coord

[0028] In one embodiment, the tessellation engine 100 was enabled by selecting the tessellation engine path in a output path control register, indicating either continuous, discrete or adaptive. The tessellation engine also specifies a maximum tessellation level. For adaptive tessellation mode, this is the maximum tessellation clamp value. For continuous and discrete tessellation mode this is a tessellation level. For the adaptive tessellation mode, there is a minimum tessellation clamp value. For continuous and discrete tessellation loads, this register is not applicable in this embodiment of this present invention.

[0029] FIG. 3 illustrates an exemplary embodiment of a data field 200 which may be stored in the memory 116 of FIG. 1. The data 200 includes a tessellation factor field 202 storing three tessellation factors 204 for a triangle primitive type and an original indices field 206 storing original indices 208. The data field 200 further includes a base address 210 for reference of the

tessellation factor therefrom. The tessellation factor 202 and the original indices 206 may have a data size relative to the corresponding number of vertex shaders 114 of FIG. 2 such that the associated tessellation information may be generated therefrom. It is recognized by one having ordinary skill in the art and any other suitable memory data structure may be implemented and that the data structure 200 of FIG. 3 is for illustration purposes only.

[0030] FIG. 4 illustrates the steps of the method for dual pass adaptive tessellation. The method begins, step 220, by receiving vertex information and an index list, one list is received from a memory device in step 222. With reference to FIG. 1, index list 120 is retrieved from a memory interface across a direct memory access engine 108. The next step, step 224, an auto-index value for each of the primitive. Also from the first pass, the next step is generating a plurality of shader sequence outputs, 226. Illustrated with reference to FIG. 1, the plurality of shader sequence outputs 126 are provided to in plurality of vertex shader input staging registers 112, step 228. Therefore, within the first pass, a plurality of tessellation factors are generated in response to shader sequence output, step 230.

[0031] The shader sequence output 136, in one embodiment, is stored in a memory location such that they may be further received as a plurality of indices, step 232, during a second pass. As illustrated in FIG. 2, the tessellation factors 150 are provided to the memory interface 102 such that they may be retrieved from the direct memory access engine 108. As such, one embodiment of the method is complete, step 234.

[0032] FIG. 5 illustrates another embodiment of a method for dual pass adaptive tessellation. The method begin, 250, by performing the steps during a first pass of the steps 222 through steps 230 of FIG. 4, step 252. As discussed above, the steps include receiving primitive

information and index list, and an auto-index value for each of the primitive, generating a plurality of shader sequence output, providing the shader sequence output to a plurality of vertex shader input staging registers and generating a plurality of tessellation factors in response to the shader sequence outputs. The next step, step 254, is writing the plurality of tessellation factors to a memory device. As discussed above with regards to FIG. 1, the memory 116 may be any suitable memory device capable of storing and allowing for the retrieval of the tessellation factors therefrom. Steps 252 and 254, in the present embodiment, occurred during a first pass of the tessellation engine 100 of FIGS. 1 and 2.

[0033] During a second pass, the first step, 256, is receiving the tessellation factors from the memory device. The next step, step 258 is generating an auto-index value for each of the plurality of indices. The next step, step 260, is generating a plurality of bary-centric coordinates based on the tessellation factors, or in another embodiment, tensor product coordinates may be computed based on primitive type. As discussed above, the coordinates may be (u, v) or (u, v, w) based on the higher order surface.

[0034] Still within the second pass, the next step, step 262 is computing a plurality of tessellated vertices by fetching a control point specified by the auto-index value for each of the plurality and indices. In one embodiment, the control point specified by the auto-index value for each of the plurality of indices may be disposed within the memory 116 illustrated in FIG. 1 and provided to the vertex shaders 114. As such, output signals 130, 132 and 134 generated by the vertex grouper tessellator 106 of FIG. 1 and sub primitive data further generated by the vertex shaders 114 of FIG. 2.

[0035] It should be understood that there exists implementations of other variations and modifications of the invention and its various aspects, as may be readily apparent to those of ordinary skill in the art, and that the invention is not limited by the specific embodiments described herein. For example, the number of vertex shaders and vertex shader instruction staging registers may be adjusted based on the number of tessellation factors to be computed and the shader processing unit output may also be adjusted to correspond to the number of vertex shaders and vertex shader instruction staging registers. It is therefore contemplated and covered by the present invention any and all modifications, variations, or equivalents that fall within the scope of the basic underlying principles disclosed and claimed herein.